📋 API CA Prep Day 1: Models and Service

Table of Contents

1. Core Requirements

You are tasked with building the foundation for an Event Planner API. This includes setting up the database models, configuring Sequelize, writing a service layer, and writing unit tests for the service logic.

Project Setup:

1. Create a new Express application using the **Express Generator** (without views):

   ```
   express events-api --ejs
   ```

   NOTE: Even though we are making an API and should have no views, the starter tempalte does, so we will follow its structure.

2. Install the required packages:

   ```
   npm install sequelize mysql2 dotenv
   npm install --save-dev jest
   ```

3. Set up `.env` to store your database credentials:

   ```
   DB_USER=root
   DB_PASS=password
   DB_NAME=events
   DB_HOST=localhost
   DB_DIALECT=mysql
   ```

2. Models

EventType Model

Field	Type	Constraints

Field	Type	Constraints
id	INT	Primary Key, Auto Increment
name	STRING	Required, Unique

Event Model

Field	Type	Constraints
id	INT	Primary Key, Auto Increment
title	STRING	Required, min length 3
date	DATE	Required, must be future date (custom validator)
location	STRING	Optional
eventTypeId	INT	Foreign Key → EventType(id), Required

Relations:

- Event belongs to EventType.
- EventType has many Events.

Sequelize Configuration:

- Inside `/models` folder:
  - Create `index.js` to configure Sequelize:
    - Load models dynamically.
    - Export Sequelize instance + models.
  - Move DB config to `models/dbConfig.js`:

```
module.exports = {
  username: process.env.DB_USER,
  password: process.env.DB_PASS,
  database: process.env.DB_NAME,
  host: process.env.DB_HOST,
  dialect: process.env.DB_DIALECT,
  define: {
    timestamps: false
  }
}
```

3. Test Data

In your **test file**, after syncing the database (in the `beforeAll` hook - see point 5 for details), hardcode using `bulkInsert`:

- **4 EventType records**:

```
[
  { id: 1, name: "Conference" },
  { id: 2, name: "Meetup" },
  { id: 3, name: "Workshop" },
  { id: 4, name: "Seminar" }
]
```

- **2 Event records**:

```
[
  { id: 1, title: "Test Event 1", date: "2025-05-01", location: "Oslo",
eventTypeId: 1 },
  { id: 2, title: "Test Event 2", date: "2025-06-10", location: "Bergen",
eventTypeId: 2 }
]
```

4. Service Layer

Create a **services/EventService.js** file:

- Class: `EventService`
- Constructor receives Sequelize `db` object.
- Required Methods:
  - `create(data)`
  - `getById(id)`
  - `getAll()`
  - `update(id, data)`

5. Testing

> **NOTE**: You are not required to implement service level tests for the CA, but its worth learning how it can be done.

Test Setup:

- Configure Jest.
- Mock `models/dbConfig.js` to connect to a test MySQL database named `testEvents`.

```
jest.mock('../models/dbConfig', () => ({
// Mocked config (username, etc.) goes in here
}))
```

  TIP: You can add `logging: false` to not have sequelize bloat your test output. Be aware you will have to find other ways to debug if things go wrong (or just turn on logging until its sorted)
- Sync the DB and insert the hardcoded EventType and Event records inside `beforeAll()`.

Write Unit Tests for EventService:

Test Case	Expected Outcome
Get all events	Returns array of events
Get event by ID (valid)	Returns correct event
Get event by ID (invalid ID)	Returns null
Add event (valid data)	Creates event successfully
Add event (invalid data: title/date)	Throws ValidationError
Add event (invalid EventTypeId)	Throws ForeignKeyConstraintError
Update event (valid ID & data)	Event updated (returns 1 row affected)
Update event (non-existent ID)	Returns null (not 0 rows affected)
Update event (invalid data)	Throws ValidationError